Directed Placement for mVLSI Devices

BRIAN CRITES, KAREN KONG, and PHILIP BRISK, University of California, Riverside

Continuous-flow microfluidic devices based on integrated channel networks are becoming increasingly prevalent in research in the biological sciences. At present, these devices are physically laid out by hand by domain experts who understand both the underlying technology and the biological functions that will execute on fabricated devices. The lack of a design science that is specific to microfluidic technology creates a substantial barrier to entry. To address this concern, this article introduces Directed Placement, a physical design algorithm that leverages the natural "directedness" in most modern microfluidic designs: fluid enters at designated inputs, flows through a linear or tree-based network of channels and fluidic components, and exits the device at dedicated outputs. Directed placement creates physical layouts that share many principle similarities to those created by domain experts. Directed placement allows components to be placed closer to their neighbors compared to existing layout algorithms based on planar graph embedding or simulated annealing, leading to an average reduction in laid-out fluid channel length of 91% while improving area utilization by 8% on average. Directed placement is compatible with both passive and active microfluidic devices and is compatible with a variety of mainstream manufacturing technologies.

CCS Concepts: • **Applied computing** → **Health informatics**; • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Hardware** → **Placement**; **Emerging architectures**; **Biology-related information processing**; **Microelectromechanical systems**;

Additional Key Words and Phrases: Microfluidics, directed placement, mVLSI

ACM Reference format:
Brian Crites, Karen Kong, and Philip Brisk. 2019. Directed Placement for mVLSI Devices. *J. Emerg. Technol. Comput. Syst.* 16, 2, Article 14 (December 2019), 26 pages.
https://doi.org/10.1145/3369585

14

1 INTRODUCTION

Microfluidic chips are poised to revolutionize biochemistry and bioengineering through automation, miniaturization, and programmability. The ability to precisely control the volumes of expensive reagents at the microliter scale and below has enabled relevant biological applications such as single-cell capture [7] and protein analysis [60] and significantly increased the throughput of multiple important laboratory functions [4, 15, 21, 59].

While a handful of large and well-funded academic laboratories possess both the engineering and biological expertise to design, fabricate, test, and validate microfluidic chips as prerequisites for using them to produce publishable advances in the biological sciences, the vast majority of

This work was supported in part by NSF Awards #1351115, #1545097, #1740052, and #1910878.
Authors' addresses: B. Crites, K. Kong, and P. Brisk, University of California, 900 University Ave, Riverside, CA 92521; emails: {bcrit001, kkong006}@ucr.edu, philip@cs.ucr.edu.

1550-4832/2019/12-ART14 $15.00
https://doi.org/10.1145/3369585

laboratories lack the requisite expertise. Today, most advances in microfluidics are generated by engineering-oriented laboratories; meanwhile, the majority of biological research laboratories do not produce their own microfluidic solutions, which limits the type of research problems they can explore. A second limiting factor is cost: starting with Stanford's gas chromatographic air analyzer in 1979 [48], researchers have relied on semiconductor-inspired microfabrication techniques to produce the majority of microfluidic devices [3, 13, 34, 54, 61]. This necessitates expensive clean rooms, fabrication equipment, and highly trained technicians, which is once again prohibitive for nontechnologists.

There is an urgent need to increase access and lower barriers to entry for researchers who would like to integrate microfluidic solutions into their laboratories. The recent rise of low-cost rapid prototyping technologies, namely desktop CNC milling [23, 24] and 3D printing [10, 43, 56], addresses the equipment and facilities barriers; however, they do not solve the key engineering design challenge. At present, most microfluidic chip designers use general-purpose CAD software such as AutoCAD or SolidWorks, which lacks domain knowledge. Using semiconductor hardware design as an analogy, it is certainly possible to use SolidWorks or AutoCAD to produce the photomask set that defines the geometries used during the photolithography steps of semiconductor fabrication; however, doing so is thoroughly impractical due to the presence of software produced by leading companies in the Electronic Design Automation (EDA) industry, namely Cadence, Synopsys, and Mentor, which are tailored to the needs of semiconductor designers. The creation of similar software specialized for microfluidics could substantially simplify the design process. The combination of low-cost desktop fabrication equipment and easy-to-use design software will substantially improve accessibility to microfluidic technologies in both research and education.

One important algorithmic step for microfluidic design automation software is physical design, i.e., placement and routing. While many microfluidic chips visually appear to be similar to semiconductor chips, the physical design constraints are notably different. The two key differences are the lack of a fluidic analog to both multilayer metallization and standard cells. The former limits fluid processing and transport functions to a single device layer, while the latter means that physical design tools must be able to place components having arbitrary geometries. Consequently, the design rules that govern placement and routing are substantially different for microfluidic layout in comparison to semiconductor physical design. Using another semiconductor industry metaphor, the current design paradigm for microfluidics is stuck in the early 1970s, before the Conway-Mead revolution led to integrated semiconductor VLSI technology and computer-aided design tools, which are now industry standard [33], which limits the design and integration complexities that can be achieved by manual layout.

Contribution. Most microfluidic devices have a naturally *directed* structure: fluid is injected into the device via designated input ports, flows through the devices for process, and exits the device via designated output ports. Based on this observation, this article introduces *Directed Placement*, an efficient and effective heuristic for microfluidic physical design. Directed placement yields layouts that share many principle similarities to designs that are produced by hand, which makes the designs easier to reason about if or when modifications are needed. Quantitatively, Directed Placement yields shorter channel route lengths than several existing placement heuristics when used in conjunction with existing routing methods. Specifically, we compare against three previous microfluidic placement methods: Simulated Annealing [29], which was adapted from semiconductor VLSI placement; Planar Placement [30], which adapts planar graph layout methods for microfluidics; and Diagonal Component Expansion [6], which is a more refined variant of the Planar Placement method. Directed Placement is shown to be far more effective than three prior heuristics in terms of improving area utilization and reducing average channel length.

2 MICROFLUIDIC TECHNOLOGY OVERVIEW

This section provides an overview of microfluidic fabrication technologies and the ways that they can be used to fabricate *passive* and *active* devices. Passive devices do not contain any active elements (e.g., valves) and, as such, are simpler, cheaper, and easier to fabricate than active devices and require less external equipment to operate. In principle, Directed Placement can lay out both passive and active devices. We discuss both device types here but will focus our examples on passive devices as they are the most restrictive and aid in explanation clarity.

2.1 Fabrication Technology

Microfluidic devices rely on a continuous flow of fluid through a network of microchannels and components that are patterned onto one or more device layers. Each patterned layer may be a rigid substrate [8, 19, 38] or imprinted in a flexible polymer (e.g., *polydimethylsiloxane (PDMS)*) [61]. An additional layer of material, which is typically rigid and *not* patterned (e.g., a glass slide), is bonded to the topmost patterned layer to enclose the channel network, which would otherwise be exposed to the environment. Depending on the specific choice of materials used, small holes may be drilled (rigid material) or punched (flexible material) into the second layer to provide I/O access for fluids.

The process to create each patterned layer is technology dependent. In general, more expensive fabrication equipment is necessary to produce patterns with smaller features; that said, many applications in biology are not compatible with features smaller than the biological media being studied (e.g., cells, DNA molecules, etc.). In terms of rigid substrates, desktop CNC milling [23, 24] represents the lower-cost, larger-feature-size end of the spectrum, while microfabrication represents the higher-cost, smaller-feature-size end of the spectrum. CNC milling typically cuts patterns into polycarbonate thermoplastic polymers, while microfabrication etches patterns into glass via photolithography. In certain cases, compatibility between the device material and biological media under study may dictate the choice of fabrication technology as well.

Imprinting patterns in flexible materials is more complicated and expensive than patterning a rigid material [61], as a photolithographic process is required to produce physical molds. The flexible material must be available initially in a liquid form so that it can be poured onto the mold. While resting on the mold, the liquid must partially harden to imprint the mold's pattern, before it can be removed and mounted on a rigid substrate. A number of additional challenging steps (e.g., degassing to remove air bubbles) are necessary as well, which increases the time and cost of this fabrication process while inhibiting scalability. Further details are omitted to conserve space.

Additive manufacturing (e.g., 3D printing [10, 43, 56]) can create enclosed rigid 3D structures. The key advantage of 3D additive manufacturing is the elimination of a separate bonding step between distinct patterned and enclosing material layers. The most significant disadvantages are twofold: first, unlike glass or PDMS, 3D printed objects are opaque, which makes the material a poor choice for use in biological studies that involve imaging; second, there is concern about the biological toxicity of 3D printed parts; recent research has shown that different 3D printing technologies yield parts with different toxicity levels and that exposure to ultraviolet light largely mitigates these issues in the most extreme cases [37].

Directed Placement primarily targets 2D channel networks; the authors are unaware of any 3D physical design algorithms targeting 3D printed microfluidics. The subsequent discussion will focus exclusively on devices created by one or more 2D patterned device layers.

2.2 Passive Devices

Passive microfluidic devices rely primarily on the underlying physical properties of fluid flow to achieve their desired microfluidic functionality [11, 57]. A passive microfluidic device has

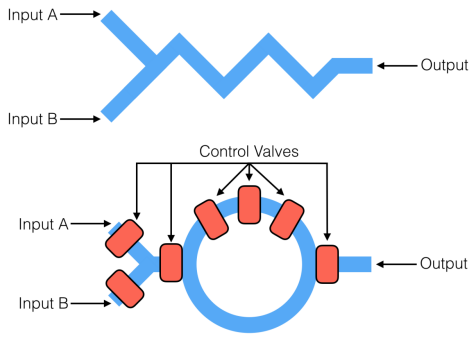

Fig. 1. A passive serpentine mixer (top), the shape of which allows for dilution between the two fluids, causing the two input flows to mix, and a rotary mixer (bottom), which uses peristaltic pumping to mix.

no moving parts (other than the fluid or fluidics that flow within it). Each component in the channel network is designed to ensure that it can perform a specific action, such as the serpentine channel mixer, shown in Figure 1, which allows dilution to mix two flows of fluid together. The only physical connections to a passive device are fluid inputs and outputs. Driven by an external pressure source such as a syringe pump, fluid flows into the device, through the device to perform its function(s), and then out of the device into an external reservoir to collect the excess.

In terms of technology, a passive microfluidic device typically comprises two material layers: a *patterned* layer, which may be rigid or flexible (see the preceding subsection), and a *nonpatterned* enclosing layer, which is typically rigid (e.g., a glass slide). Once the two layers have been mounted and bonded, the final fabrication step is to create holes that will act as the fluidic I/O interface. If the patterned layer is rigid, the holes are drilled; if it is flexible, the holes are punched. I/O holes do not penetrate the enclosing layer.

2.3 Active Devices

Active microfluidic devices employ pneumatically controlled microvalves to actuate transport and mixing of fluids [3, 13, 34, 54]. Microvalves typically perform one of two functions: active pumping and mixing, or reconfiguring the various fluidic pathways through the device. While beyond the scope of this particular article, it is worth noting that several research groups have shown that microvalves introduce an element of programmability into microfluidics that is not otherwise achievable in passive devices [9, 55].

Active microfluidic devices are more complicated and expensive to design, fabricate, and test when compared to passive devices and require additional external equipment to deliver pneumatic control signals. Active devices typically require multiple patterned substrate layers, at least one of which uses a flexible material, which acts as a membrane to produce pneumatically controlled microvalves [3, 13, 34, 54].

Active devices typically are partitioned into "fluidic" and "pneumatic" layers. The fluidic layer performs fluid transport and operations that directly interact with the fluid. The pneumatic layer, in contrast, is effectively a channel network that delivers pneumatic control signals, provided by an external source, to the (set of) microvalve(s) that each control input drives.

2.3.1 Multilayer Soft Lithography. The first widely recognized and successful microvalve technology was based on *multilayer soft lithography* [54]. In single-layer soft lithography [61], a channel network is patterned into PDMS (once again, a flexible material) and then mounted on a glass

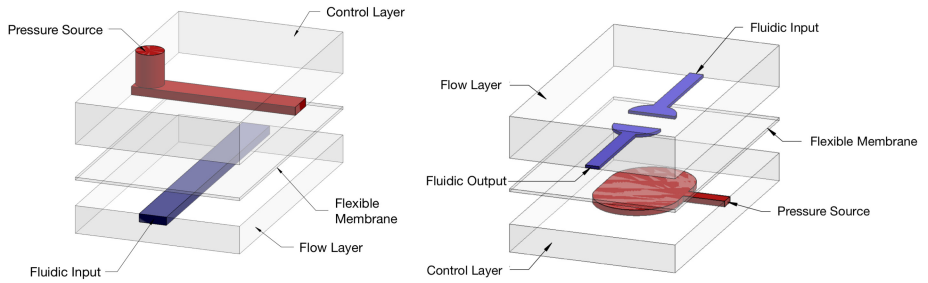

Fig. 2. Exploded views of (a) multilayer soft lithography and (b) monolithic membrane valves.

slide as an enclosing layer. Multilayer soft lithography employs a similar principle but stacks two or three imprinted PDMS layers on top of the rigid substrate.

Figure 2(a) shows an example of a two-layer multilayer soft lithography microvalve: the *flow layer* on the bottom manipulates biological fluids of interest, while the *control layer* above provides actuation capabilities from the external pressure source. The microvalve is formed where a control channel on one top layer crosses a flow channel on the bottom layer. By default, all microvalves are open; pressurizing a control channel closes the microvalves that it drives. The control channel geometry at the microvalve point must be wide enough to allow a small amount of external pressure to deflect the small PDMS membrane to close the microvalve.

Similar to transistors or logic gates, microvalves can be combined to form larger components such as peristaltic pumps, mixers, and multiplexers, which can then be integrated to form fully integrated microfluidic devices, akin to integrated semiconductor circuits [34]. Figure 2 (bottom) shows a rotary mixer. Valves at the mixers' input and output allow two fluids to be loaded and for the mixer to be isolated off from the rest of the device during mixing; when the mixer is closed, the three-valve pump applies peristaltic actuation to actively mix the fluids. The mixed fluids can then be driven out of the mixer at its output.

Since its inception in the year 2000, the integration density of microvalves has followed a trend similar to Moore's Law under multilayer soft lithography [14]; this was called *microfluidic Large-Scale Integration (mLSI)*. In 2012, microvalve densities between 0.4 and 0.8 valves per cm^2 were reported [3], which represented an increase in valve density of more than two orders of magnitude; this led to the terminology *microfluidic Very Large-Scale Integration (mVLSI)*.

2.3.2 Monolithic Membrane Valves. While highly successful from a technological standpoint, multilayer soft lithography has a high barrier to entry due to the cost and complexity of imprinting channel network patterns onto PDMS. A secondary concern regarding PDMS is its nonspecific absorption, as a porous material, which limits its use in many potential biological applications [18, 46].

To address these concerns, an alternative *monolithic membrane valve* was introduced in 2003 [13]. As shown in Figure 2(b), a monolithic membrane valve consists of a thin unpatterned PDMS membrane sandwiched between two patterned rigid layers. Fluid flows in one of the rigid layers, allowing the PDMS membrane to act as the enclosing layer. The second layer delivers external pressure to each microvalve. Monolithic membrane valves are normally closed, and can be opened by applying vacuum or pressure via the control layer.

Monolithic membrane microvalve technology has not achieved the same integration densities as multilayer soft lithography; however, its key advantage is cost. The initial monolithic membrane valve designs employed etched glass as the rigid substrates [13], which was considerably simpler

and more cost effective than patterning PDMS; moreover, providing a rigid, rather than porous, channel wall on three out of four sides mitigated the absorption issue. Subsequently, the same basic monolithic membrane microvalve design has been translated to both 3D printing [43] and CNC milling [24]; while 3D printing and CNC milling cannot achieve feature sizes as small as glass etching, they are much cheaper and cost effective, making them far more accessible to researchers in the life sciences.

2.4 Microfluidic Technology: Summary

Active microfluidics, and multilayer soft lithography in particular, have achieved tremendous academic impact, and the underlying technologies show great promise to integrate semiconductor-like complexity into biological instrumentation. That said, to date, the vast majority of microfluidic devices that are produced commercially today are passive. The key driving factors are cost and complexity. In addition to fabrication costs, which we discussed previously, other cost factors include multilayer assembly, testing, and validation (e.g., to ensure that layer alignment errors during assembly did not occur), and the cost and complexity associated with external solenoid pressure sources and software, which are needed to operate the device. The extra equipment required to run the device may prohibit its use in the field, e.g., for point-of-care diagnostics.

Additionally, while microvalve integration density has increased over time, the integration density of the external control capabilities has not. For example, the Stanford Microfluidic Foundry limits the number of I/Os for both fluid and control to 35 as a design rule, regardless of the number of integrated microvalves to be controlled [1]. This limits the ability to harness the benefits of highly integrated valve densities outside of lock-step SIMD-style parallel control patterns.

All of the aforementioned issues have informed our microfluidic physical design strategy.

3 RELATED WORK

Physical design for continuous flow microfluidics differs substantially from modern semiconductor physical design in several key respects. Nowadays, semiconductor physical design is built on a foundation that includes standard cells, larger IP blocks, multilayer metallization, and parameterized geometric design rules, the origins of which are generally attributed to Conway and Mead, who introduced the notion of VLSI in the late 1970s [33].

For a typical microfluidic chip, the fluid flow layer (both components and channels) must be planar, and components may have arbitrary geometries. This is quite different from standard cell design methodologies used in semiconductor VLSI, in which cells are placed in rows with uniform height: the placement of a standard cell is characterized by a tuple (r, x), where r is the ID of the row into which it is placed and x is the horizontal offset from the leftmost position in the row. In contrast, the placement of a microfluidic component is characterized by an (x, y) pair, which represents the position of the component in a 2D plane. For active devices, additional steps (and optimization opportunities) are needed for control layer physical design.

3.1 Planarization

As described above, the flow layer of a typical microfluidic chip is limited to a single device layer. In other words, only planar netlists can be placed legally, given this design constraint. For passive devices, this imposes a planarity test on the netlist: a nonplanar netlist, which is any netlist that contains one or more subgraphs isomorphic to one of the Kuratowski subgraphs shown in Figure 3, cannot be placed legally as proven by Kuratowski [22]. One possibility is to redesign the device, if possible, in a manner that ensures planarity. Another option may be to switch to a fabrication technology, such as 3D printing, that can admit nonplanar devices. The latter option is an interesting potential research topic but goes far beyond the scope of this article.

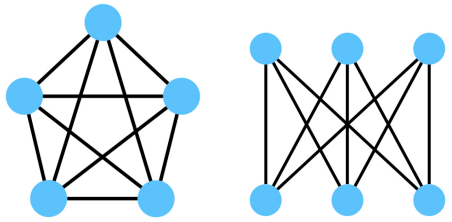

Fig. 3. The Kuratowski subgraphs (a) K_5 and (b) $K_{3,3}$. If a subgraph is isomorphic to one of these, then the entire graph is known to be nonplanar.

Active devices can be planarized algorithmically by inserting valve-based switches at points where two or more flow channels cross. As a design rule, the Stanford Microfluidic Foundry limits the number of I/Os for both fluid and control to 35 [1]. Many of these I/Os will be consumed by the netlist itself (fluid I/O) and for valve actuation. As a separate constraint, many devices that are used for biological experiments in the field or as point-of-care diagnostics in resource-limited settings may have more stringent I/O constraints as well.

Planarization for active devices has been proposed as a netlist preprocessing step prior to physical design [53]. An alternative approach is to allow a limited number of planarity violations during physical design [32], as long as the total I/O constraint is not violated. Directed Placement assumes that the netlist is planar prior to layout, which is sufficient for both active and passive devices.

3.2 Flow Layer Physical Design

Directed Placement is most similar to several prior heuristics that start with a planar graph embedding and one-by-one expand vertices (initially points) into 2D components, shifting the positions of yet-to-be-expanded components to prevent overlap and to preserve planarity [6, 30]. We compare directly to *Planar Placement* as well as the similar but largely more effective *Diagonal Component Expansion (DICE)*. We report substantial improvements in area utilization and fluid channel length; both DICE and Directed Placement employ the same fluid channel router.

We also compare Directed Placement to a simulated annealing/based microfluidic placer [32, 35], which does not guarantee planarity; planarity can be achieved after physical design via switch insertion, as described above. Directed Placement reports improved utilization and fluid channel length, despite the fact that Directed Placement imposes a planarity constraint while the simulated annealing/based placer does not, causing it to serve as a more optimal baseline for placement. We eschew additional comparisons with other flow layer placement heuristics that cannot guarantee planarity [50, 58].

We are aware of one other standalone flow channel router that could potentially compete with the one used by Planar Placement and DICE and the Directed placement heuristic presented here [27]. This router employs a heuristic that tries to simultaneously minimize total routed channel length and the length of the longest routed channel; however, it cannot ensure a planar routing result, even for planar netlists, and makes no attempt to minimize the number of channel crossings. As such, we do not consider it to be a good basis for comparison.

It is also possible to formulate flow layer placement together with routing in a single problem formulation, for example, as a Boolean Satisfiability (SAT) problem [12]. While a SAT solver can solve the joint problem optimally, it necessarily suffers from high runtimes and scalability challenges, under the assumption that $P \neq NP$. While it is possible to prune the size of the search space, e.g., by downscaling component sizes or partitioning the netlist into independent

subproblems, doing so sacrifices optimality. This points toward the possibility of investigating hierarchical partitioning and placement strategies, similar in principle to UCLA's Dragon standard cell placer [47]; however, this potential research direction is far beyond the scope of this article.

Prior work has also investigated physical design for capillary electrophoresis (CE) microfluidic chips [16, 39], which are used to perform chemical separations. The earliest approach formulated minimum-area CE placement as a nonlinear program [39], which lacks scalability; the routing stage adjusts the placement and inserts I/Os after the fact and on the perimeter of the chip, sacrificing optimality. A more recent and more effective approach [16], which reduced area compared to the former, applies simulated annealing to minimize area and routing cost, followed by an auxiliary routing step to minimize total channel length and the number of bends, and I/O placement to minimize the length of auxiliary channels. The most fundamental difference compared to our work is that Directed Placement treats I/Os as components, treating them as part of the netlist, as opposed to placing and routing them as a postprocessing step. Ignoring this difference, Directed Placement could be used in lieu of the simulated annealing step. The simulated-annealing/based placer that we compare against is quite similar to this approach as well.

3.3 Control Layer Physical Design

One approach to the physical design of active microfluidic devices is to first place and route the flow layer, e.g., using the techniques described in the preceding subsection, and then generate the control layer afterward. Control layer generation necessitates the placement of control inputs and control layer routing, which connects each control input to the valve (or valves) that it drives.

Valve sharing typically occurs in one of two contexts, noting that the total number of I/Os is limited as a design constraint. In the first context, microfluidic chips are designed to process fluids in a lock-step SIMD fashion [60], and this design choice is reflected as an inherent property of the netlist; this allows one set of valves to control k independent or mostly independent datapaths, which increases throughput without increasing demand for control inputs.

In the second context, valve sharing is posed as an application-specific optimization: two valves whose actuation timings are wholly independent from one another can share the same control input [2, 36]. In this case, the overall objective is to minimize the total number of control inputs, which reduces total chip area and increases the likelihood of satisfying design constraints. Another strategy to reduce the number of control inputs is to instantiate a large control demultiplexer on the perimeter of the chip [9, 52, 64]: this allows n external control inputs to independently drive 2^n internal control lines; the drawback is that the control demultiplexer and associated control routing area itself may be quite large, and in certain cases, it dominates the overall chip area. To reduce the cost, the demultiplexer, which is a general-purpose solution, can be replaced with application-specific pneumatic control logic [20], which can be optimized to minimize gate count [41] or to enhance testability [28, 40].

Valve sharing can be performed prior to control layer physical design or as an integrated co-optimization step [17] with control placement and routing algorithms. One advantage of the integrated approach is that, to achieve precise timing, it may be necessary to perform (near-)equal length routing, so that the application of pressure at a control input actuates all of the valves that are driven at precisely the same time [62].

With respect to this article, control layer generation (with or without valve sharing) is performed *after* Directed Placement generates the flow layer; as such, they are complementary steps that are only applicable to the automated layout of active microfluidic devices.

3.4 Combined Flow/Control Layer Physical Design

It is also possible to simultaneously generate the flow and control layers of an active microfluidic device, which creates opportunities for co-optimization. One early effort in this direction [63]

generated the flow layer using planar placement, improved it using simulated annealing, and then generated the control layer using A* routing (valve sharing and other control optimizations were not considered). The algorithm adjusts the flow layer layout if control layer congestion exceeds a threshold. In principle, Directed Placement could be used to generate the initial flow layer.

The Columba [53] and Columba 2.0 [51] frameworks formulated this problem as an Integer Linear Program (ILP), which could be solved optimally, albeit in exponential worst-case time under the assumption that $P \neq NP$. Because of these scalability issues, the same authors introduced Columba S [52], which severely restricted the design space. The flow layer restrictions include the following: (1) all inputs are on the left side of the chip; (2) all outputs are on the right side of the chip; and (3) all flow channels are horizontal and do not bend. Despite the ILP, this problem is actually more restricted than what we propose for Directed Placement. The third limitation necessitates the introduction of additional switches in certain cases, an issue that does not affect Directed Placement (when placing and routing active devices).

The Columba S control layout includes several nice features, including multiplexers at the top and/or bottom of the chip to reduce the number of control inputs, and the ability to route through and bypass components; this latter feature could be integrated into other physical design algorithms as well. The one key restriction on control layer generation is that all control lines are vertical (except at component bypass points). While the Columba S ILP formulation simultaneously generates the flow and control layers, it would just as easily be possible to start with a flow layer generated by Directed Placement and use a reduced version of Columba S to generate the control layer.

4 PRELIMINARIES

An mVLSI netlist $M = (C, E)$ consists of a set of components, C, and a set of edges, E, between them. A component $c_i \in C$ is a tuple $c_i = (T_i, P_i, x_i, y_i, h_i, w_i)$, where T_i is the set of neighboring components to c_i, P_i is the set of c_i's ports, (x_i, y_i) is the coordinate location of the upper left corner of c_i, and h_i and w_i are the height and width of c_i, respectively. A port on a component $c_i \in C$, $p_{i,j} \in P_i$ is located at $(a_{i,j}, b_{i,j})$, a point on the perimeter of c_i; c_i is called a *terminal component* if $|T_i| = 1$. An edge, $e_i \in E$, is a pair of components $e_i = (c_i, c_j)$, which represents a fluidic connection between them. An optional set of components $I \subset C$ can also be provided that represents the inputs of the microfluidic device.

A lane L_i is defined to be an ordered set of components that align vertically. These lanes are numbered and ordered $L_0, L_1, \ldots L_n$, where L_0 is the left (west)-most lane and L_n is the right (east)-most lane. The first component in the set $c_0 \in L$ is the top (north)-most in the lane and the last component in the set $c_{|L|-1} \in L$ is the bottom (south)-most in the lane. Adjacent lanes may be separated by an optional buffer space Δ to improve routability and/or to satisfy fabrication design rules relating to spacing.

5 PLACEMENT

5.1 Preprocessing

Directed Placement uses a microfluidic netlist as an input but does not require a microfluidic application in order to perform placement and routing. Because no application is given as input, no optimizations can be made to the netlist, since it would be impossible to determine if a change to the netlist would render the application unable to map. Previous methods for generating and optimizing netlists based on applications [35] have been proposed, and methods to optimize the netlist before placement and routing are compatible with the Directed Placement method. For these reasons, architectural optimization is out of the scope for this work, and the assumption is made that all components and connections are required to create a valid layout.

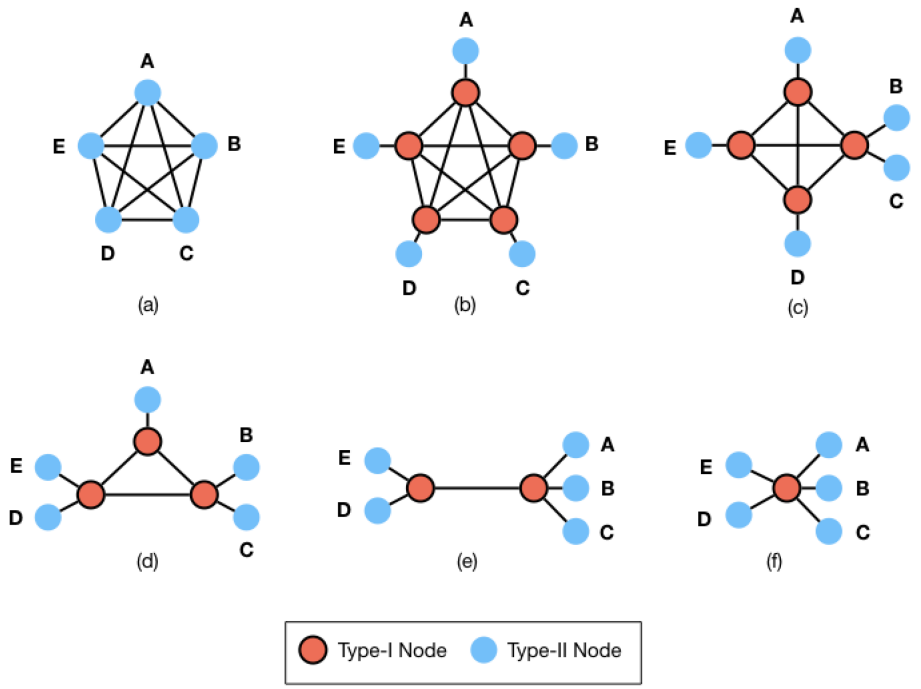

Fig. 4. (a) The input graph is converted such that (b) Type II nodes are introduced for all components with an edge degree larger than 2 and connected to the original Type I nodes. (c–f) Pairs of Type I nodes that are then connected through an edge are iteratively combined until no more pairs exist. In the case of the K_5, this results in a single Type I node that will be replaced with a five-way switch.

Directed Placement requires that the input device architecture is planar, as this is a requirement for the manufacturing of the physical device. Planarity in a graph can be determined by the absence of the Kuratowski subgraphs K_5 and $K_{3,3}$ (illustrated in Figure 3) as proven in Kuratowski's theorem [22]. If a nonplanar graph is given as input for Directed Placement, then the planarization method introduced by Tseng et al. [53] can be used to preprocess the nonplanar input into a planar one for placement, routing, and fabrication. A short description of this method follows here for completeness.

First, a new graph of the system is constructed with two different node types. The first is a Type I node, which represents a switch that will be inserted into the system and can have an unconstrained number of edges. The second is a Type II node, which represents any component within the system and will be constrained to having a maximum of two edges. The original input architecture is then processed with Type II nodes representing each component. If a given component has more than two edges, a Type I node is introduced with all the components' original edges routing to the new Type I node along with an additional edge between the Type I node and the Type II node representing the component. After the entire input has been processed in this way, the resulting graph is then iteratively reduced by combining any two Type I nodes that connect through an edge. When all possible reductions of this type have been completed, then every Type I node left in the system is replaced with a switch component capable of handling the number of edges associated with that node and the input graph has been planarized, and each Type II node is replaced with the component it represents. A short example showing the planarization process of a K_5 subgraph can be seen in Figure 4.

It should be noted that this method requires the insertion of switches into the system that require valves to operate and can therefore only be used on active devices. Passive devices that are nonplanar cannot be fabricated onto a single layer.

5.2 Initial Lane Assignment

As an optional first step, all input components $c_i \in I$ are added to the first lane L_0. Many microfluidic devices naturally place all of the inputs on one side, and, without loss of generality, during device operation, the fluid tends to flow from one side of the device to the opposite side. In all our examples we utilize a west-to-east flow direction, but this could easily be modified by changing the orientation of the device. If I is not specified, the first step is to add the component $c_j \in C$ with the smallest $|T_j|$ to L_0. In the case that there is a tie for the smallest component, the component with the fewest ports $|P_j|$ is chosen. If there is still a tie for both the smallest component and fewest ports, then choose randomly from the available candidates.

A queue Q is created to facilitate a breadth-first traversal of the components. Initially, all components $c_j \in L_0$ are enqueued. The initial lane assignment heuristic proceeds until Q is empty.

The first step is to dequeue a new component, c_q. Each neighbor $c_r \in T_q$ that has not yet been assigned to a lane is enqueued; c_r is also assigned to lane L_{f+1}, where L_f is the lane to which c_q is assigned. If c_r is a terminal component, then it is added to L_f to allow for a short connection (c_q, c_r); we enforce the constraint that both components are placed adjacent to one another within the lane. In order to minimize the lane height and simplify the later routing, a maximum of two terminal components connected to c_q may be placed in lane L_f and all additional terminal components connected to c_q are added to lane L_{f+1}.

If an mVLSI netlist consists of multiple connected components, then some components will not be assigned to a lane once Q is empty. This is unlikely to occur when placing and routing a single microfluidic device but may occur when performing these steps for a number of different devices on a single mask in order to increase production yields for mass manufacturing. If this occurs, the unassigned component c_j with the smallest degree $|T_j|$ is inserted into Q and initial lane assignment proceeds as normal. The process terminates when all components have been assigned a lane.

Figure 5(a) depicts an mVLSI netlist, and Figure 5(b) shows the initial lane assignment after the breadth-first search completes. In Figure 5(b), components are grouped into subsets, as will be discussed in the next section.

5.3 Lane Ordering Optimization

Once each component has been assigned to a lane, those components need to be ordered within the lane to reduce route lengths. This is done by segmenting the components within a lane L_i into some number of ordered subsets $L_{i,0}, L_{i,1}, \ldots, L_{i,m_i}$ such that now the lane L_i is an ordered set of ordered component subsets, the union of which contains all the components in the original lane $L_i = L_{i,0} \cup L_{i,1} \cup \cdots \cup L_{i,m_i}$. These ordered subsets continue to form a vertical arrangement of components, with the subset $L_{i,0}$ being at the top of the lane and the subset L_{i,m_i} being at the bottom. Within these ordered subsets, the first component $c_0 \in L_{i,0}$ will be placed at the top and the last component $c_{|L_{i,0}|-1} \in L_{i,0}$ will be placed at the bottom before the next subset $L_{i,1}$ begins to be placed within the lane. There are three stages to ordering the components within their lane:

(1) **Subset Assignment:** Components within a lane L_i are assigned to a subset $L_{i,j}$ based on their neighbors in the preceding lane.
(2) **Subtree Ordering:** Components within a lane subset $L_{i,j}$ are ordered based on their subtree in successive lanes.

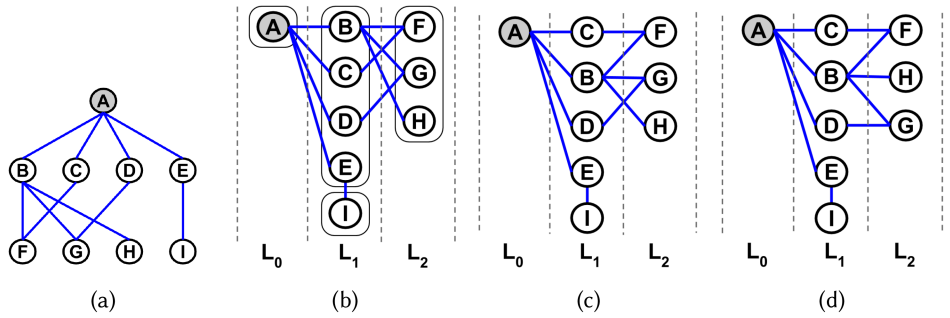

Fig. 5. (a) The mVLSI input netlist is represented as an abstract graph, with components as nodes and connections as edges. In this example, A is the only input. (b) Using a breadth-first traversal, the nodes are assigned to an initial lane based on their traversal depth. Here the different subgroups are circled for illustration. Note that I is a terminal component, so it is added to the same lane as its parent E. (c) Node B is moved to the center since its subtree $\{F, G, H\}$ is the largest. (d) In L_2, nodes F and H are added first because they are processed from their last parent in the previous lane B. G is then added because its last parent is D, which leads to a swap of G and H. This provides an abstract lane ordering but does not represent an actual placement.

(3) **Parental Reordering:** Components within a lane subset $L_{i,j}$ are reordered based on the position of their neighboring components in the previous lane.

Each step processes all components in all lanes before the next step begins.

5.3.1 Subset Assignment. In the first step, each lane L_i starting with L_0 is partitioned into subsets $L_{i,0}, L_{i,1}, \ldots, L_{i,m_i}$. In the first lane, L_0, each component $c_j \in L_i$ is partitioned into its own distinct subset such that the number of subsets $m = |L_0|$. For each subsequent lane $L_i, i > 0$, the components $c_j \in L_i$ are partitioned into subsets based on their connections to components in the previous lane L_{i-1}. All $c_j \in L_i$ connected to the same $c_k \in L_{i-1}$ are partitioned into the same subset $L_{i,s}$, where s is the lowest unused subset index in the lane L_i. If c_j connects to multiple components c_k in L_{i-1}, it is partitioned into the first possible subset. Figure 5(b) depicts the components in three lanes partitioned into subsets.

5.3.2 Subtree Ordering. The second step begins after all components $c_j \in C$ have been partitioned into some subset $L_{i,s}$. During this step, all lanes L_i and subsets within lanes $L_{i,j}$ are traversed via indices $0 \leq i \leq |L| - 1$ and $0 \leq j \leq m_i$; recall that m_i is the number of subsets in lane L_i.

First, each component $c_k \in L_{i,j}$ is sorted based on the size, measured in number of components, of its subtree in subsequent lanes $L_p, p > i$. The subtree size is determined using a breadth-first traversal starting from c_k. If the search is presently processing component c_j in lane L_b, then it is not allowed to expand to any components belonging to lane L_a where $a < b$. The number of components traversed is then used to sort the components within the lane subset, with the component with the largest subtree in $L_{i,j}$ being at the center of the subset and subsequent components being ordered away from the center. This is illustrated in Figure 5(c).

5.3.3 Parental Reordering. Once the components have been ordered based on the size of their subtree, the third step is to reorder them to remove edge crossings between lanes. When the components with the largest subtrees are moved toward the center in the previous step, doing so can increase the number of intersections between lanes. Parental reordering tries to reorder the components based on the locations of their neighbors in the previous lane (parent nodes when viewed as a tree) to remove these intersections. A new lane L_t is created to temporarily store the new

ordering of the components during the reordering. The lanes $L_{i,j}$ are iterated in reverse order from $i = |L| - 1$ to 1 and component in forward order $j = 0$ to m_i. For each component $c_k \in L_{i,j}$ from the top of the subset to the bottom, the algorithm searches through c_k's neighbors in the previous lane L_{i-1} and adds them to L_t based on their ordering in L_{i-1}. If a component in L_{i-1} is a neighbor of multiple components in L_i, then it is added to L_t when processing its last neighbor in L_i. Any components in L_{i-1} not connected to a component in L_i are then added to L_t, and the previous lane L_{i-1} is updated to $L_{i-1} = L_t$. This is illustrated in Figure 5(d).

The same steps are performed in the opposite direction, iterating the lanes from $i = 0$ to $|L| - 2$, and $j = 0$ to m_i. In this iteration, for all components $c_k \in L_{i,j}$ from the top of the subset to the bottom, we will identify neighbors in the next lane L_{i+1} and add these components to L_t, with the rest of the process continuing as previously described, and updating L_{i+1} to the ordering of L_t.

5.4 Component Rotation and Port Assignment

The previous ordering steps mean that components are in optimized locations relative to their neighbors, but it does not mean that the ports of those components are located in a good position for routing. This necessitates a component rotation step before components can be given a location and routing can be performed.

The source and sink of a connection in input architecture can be either *port assigned* or *port unassigned*. When a connection's source and/or sink is port assigned, then it is required to route to a specific port on the component it is connected to. This occurs because the component that it routes to is functionally dependent on fluids flowing into its input ports and out of its output ports. An example of this would be a rotary mixer, which requires fluids to flow in through a certain port in order for the valve actuation sequence to correctly input and mix the two fluids. When a connection's source and/or sink is port unassigned, it does not have a specific port on its source/sink component that it needs to route to and can be routed to any port that is not already port assigned. This usually occurs on components that can function in any direction equally well, such as cameras and detection mechanisms. In order to account for this, for each component $c_i \in C$, a weight is calculated for the component with rotations of 0°, 90°, 180°, and 270°, which are the only orientations that are allowed because of the grid-based routing used. For each orientation, the weight is calculated to be the sum of the number of port-assigned connections with their matching connected component in that same direction and the sum of the lesser of the number of unassigned ports or the number of connected port unassigned components. This value is calculated for each side of the component and its corresponding direction.

That is, for a component located at $L_{i,j}$, the weight in the east direction would be the number of assigned ports on the east side of the components who's connected components exist in a lane east of the component $(L_k, k > i)$ summed with the lesser of the number of unassigned ports on the east side of the component or the number of port unassigned connected components in a lane east of the component $(L_k, k > i)$. This is then calculated for the ports on the west side $(L_k, k < i)$, the north side $(L_{i,k}, k < j)$, and the south side $(L_{i,k}, k > j)$. These values are then summed to create the weight for that particular component orientation. The weight for each orientation is then calculated, and the orientation with the highest weight is chosen as illustrated in Figures 6(a) and 6(b).

Once the component has been rotated, each unassigned source and/or sink on each connection must be assigned to a port. For each component $c_u \in C$ with an unassigned port from L_0 to $L_{|L|-1}$, processing from the top of the lane to the bottom, we perform a radar sweep similar to the one described in [5] beginning in the upper left corner of the component. As the radar sweep passes components, if it sweeps past a component $c_v \in T_u$, then the associated edge $e_z = (c_u, c_v) \in E$ is processed. The Manhattan distance between each unassigned port in the source component $p_j \in P_u$ and each unassigned port in the sink component $p_k \in P_v$ is then calculated. The combination of

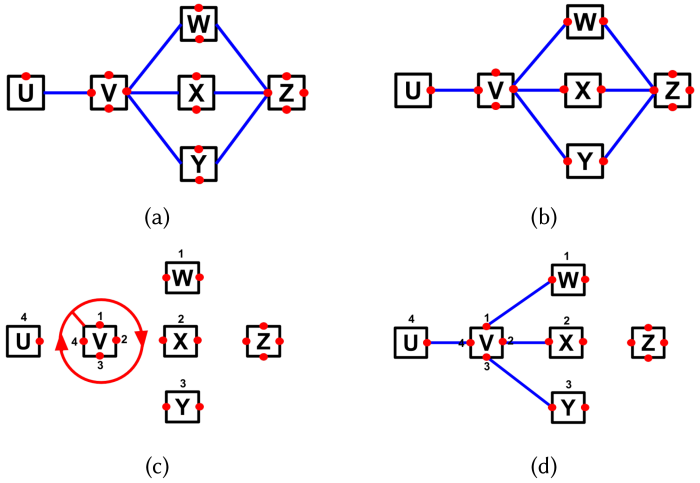

Fig. 6. (a) The mVLSI device after placement. (b) The components are rotated based on a weight function so that the number of ports facing connected components is maximized, causing U, W, X, and Y to rotate. (c, d) Port assignment is performed on component V, which performs a radar sweep to determine processing order. In this case, the Manhattan distance used for port assignment matches the radar sweep ordering.

ports with the minimum Manhattan distance is then assigned as the source and sink ports for that connection, and p_j, p_k are no longer candidate ports for later assignments, as illustrated in Figures 6(c) and 6(d). This process continues until all connections with unassigned ports have been assigned.

5.5 In-Lane Placement

At this point all components have been assigned to a lane, have been given an order within each lane, and have been rotated to optimize connection routing. However, the components have not yet been assigned an (x, y) coordinate for placement. An initial y-coordinate can be determined for each component by iterating through each lane L_i and placing the components in order, with appropriate spacing between them. The first component $c_0 \in L_i$ is given a y-coordinate of $y_0 = \Delta$ (assuming the top left of our 2D plane is the original at $(0, 0)$). This ensures that all components have enough distance from the edge of the device for routing and fabrication. Each subsequent component $c_j \in L_i$ is then placed at the position $y_j = y_{j-1} + h_{j-1} + \Delta$, which is the y-coordinate of the previous component placed shifted to account for the height of the component and an adjustable spacing quanta, Δ.

From here, the components are adjusted to better align with their neighbors in the preceding lane. The purpose is to improve routability and to try to create routes between lanes that are of similar length. For each component $c_j \in L_i, i > 0$, a new set of components $V = \{c_k \in L_{i-1} | (c_j, c_k) \in E\}$ is created. If $|V| > 1$, then the component c_j is shifted to align with the average y-coordinate of the neighboring components in V. A shift factor (δ) is calculated, such that

$$\delta = \frac{\sum_{i=1}^{|V|} y_{V[i]} + (h_{V[i]}/2)}{|V|} - y_j.$$

In the case where $|V| = 1$, V is redefined as $V = \{c_j \in L_i | (c_k, c_j) \in E\}$, and all the components in V are shifted such that the average y-coordinate of all the components in V align with the center of

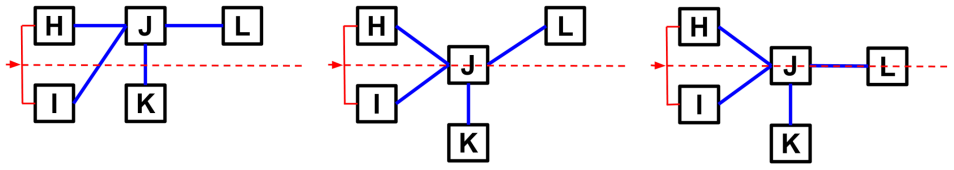

Fig. 7. (a) The horizontal dotted red line represents the vertical center of the parents of component J, calculated as the average of the y-coordinate of each parent component $\{H, I\}$. (b) Component J is shifted to the center of its neighbors in the preceding lane, shifting the other components in the lane, K, by the same amount. (c) All other components in subsequent lanes, L in this case, must also be shifted down by that amount.

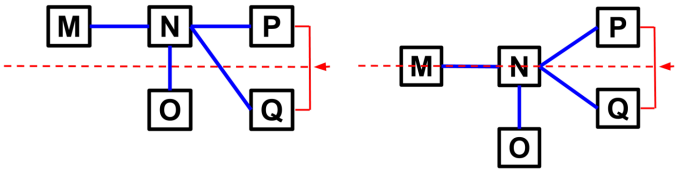

Fig. 8. (a) During a backward iteration when processing component P, the component has only a single preceding lane neighbor N. This causes the subtree of the neighbor N, which contains $\{P, Q\}$, to be shifted instead of P itself. (b) The subset $\{P, Q\}$ is shifted down to the center of their parent N. Since there are no other components in that lane and no subsequent lanes, no other components need to be shifted.

component c_k. In this case, the shift factor is calculated such that

$$\delta = \frac{\sum_{i=1}^{|V|} y_{V[i]} + (h_{V[i]}/2)}{|V|} - y_k.$$

If either c_j is shifted or the set of components in V are shifted, additional components in the lane L_i must be shifted to avoid intersections. Shifting a component c_j (set of components V) requires the movement of all the components in L_a, where $a < i$, and the rest of L_i to prevent overlap. If $\delta < 0$, we shift c_j (all components in V) upward and need to ensure no components are moved above the chip's boundaries. That is, we must maintain for each $c_t \in C$, $x_t \geq \Delta$ and $y_t \geq \Delta$. We first shift all $c_t \in C$ downward by $|\delta|$. So for each $c_t \in C$, $y_t = y_t + |\delta|$.

Finally, shift the remaining elements of L_a by δ. For each $c_t \in L_a$, where $y_t > y_j$, c_t is moved such that $y_t = y_t + \delta$. Any terminal components connected to a component c_t should also be shifted by δ. Components in L_a with $a < i$ are shifted by δ as well. At this point the set V is emptied and the process continues with the next component. Figure 7 illustrates the shifting of the single component and subsequent components, while Figure 8 illustrates the shifting of the component set.

If additional padding is required around the edge of the device to meet fabrication requirements, it can now be added. The entire device can be shifted and/or the size of the device can be increased to accommodate any padding requirements.

5.6 In-Lane Horizontal Centering

The next step is to determine each component's x/coordinate. This process begins by iterating through each lane L_i from $i = 0$ to $|L| - 1$. For the first lane L_0, all components are given an

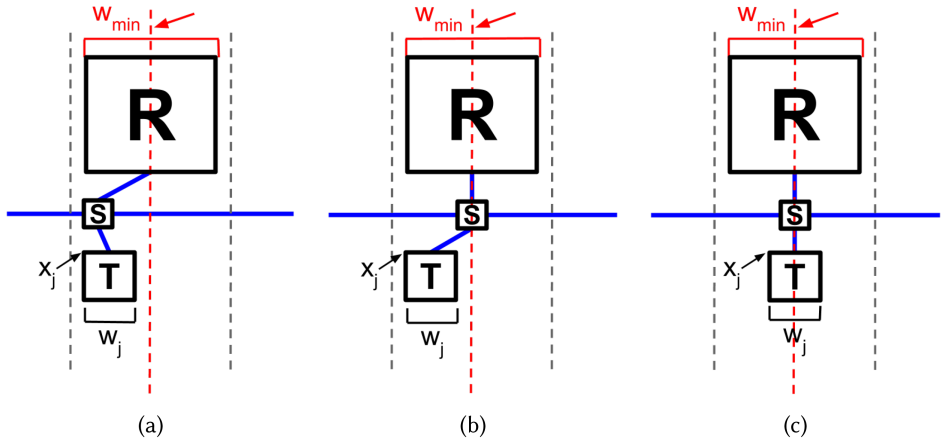

Fig. 9. (a) The vertical dotted red lines show the calculated horizontal center of the lane based on the widest component, R in this case. (b, c) The centers of component S and T are shifted to the lane center.

x-coordinate equal to the buffer space, $x_j = \Delta, \forall c_j \in L_0$. This ensures that all components in the left-most lane have enough distance from the edge of the device for routing and fabrication.

Next, the minimum width of the lane (w_{min}) is found. To prevent overlapping components between the lanes, the minimum width of the lane is equal to the component with the largest width such that $w_{min} = min(w_j), \forall c_j \in L_i$.

Once w_{min} is determined, a second iteration of all components $c_j \in L_i$ is made to center each component within the lane. Each component's x/coordinate is shifted to center the component within its lane based on the following equation, which is illustrated in Figure 9:

$$x_{j_{new}} = x_j + \frac{w_{min} - w_j}{2}.$$

Once all the components in the lane L_i have had their x-coordinate recentered within the lane, the lane iteration continues. For all lanes $L_i, i > 0$, instead of setting all components $c_j \in L_i$ initial x-coordinate $x_j = \Delta$ the initial x-coordinate is set to $x_j = x_0 \in L_{i-1} + w_{min} + \Delta$. This ensures that all the components in the next lane start far enough to the right of the previous lane to ensure there is no component overlap. Additionally, it includes enough buffer space to improve routability and meet fabrication requirements.

6 ROUTING

6.1 Flow Layer Routing

Once the components have been placed and all connections assigned ports, the routing of the connections is performed using a slight modification to the method described in [29]. A brief description of that method as well as the modifications to it is provided for completeness. A routing grid $R = (U, F)$, where U is a set of grid points and F is a set of edges representing potential channel routes between adjacent grid points. For each component $c_i \in C$, a vertex u_i for the ports $p_h \in P_i$ is instantiated and added to U. A grid of vertices is then instantiated in the empty space between components. Edges that represent potential routing channel segments are added to F by instantiating a bidirectional edge f_i with a capacity of 1 between $u_i \in U$ and $u_j \in U$ if and only if $(u_j.x - u_i.x == 1) \oplus (u_j.y - u_i.y == 1)$.

Once the grid $R = (U, F)$ has been constructed, the next step is to route channels between the components. Unlike in [30], where a network-flow-based router is utilized to do routing and port

assignment, port assignment has already been completed. Instead of a network-flow-based routing, for each port $p_j \in P_i$ of component c_i that has a connection assigned to it, a breadth-first search is made to start from the source port p_j until it reaches that connection's assigned sink port p_k. A path reclamation step adapted from Lee's algorithm [25] is then performed to find the shortest path from the sink p_k to the source p_j. The reclaimed path is then set as the final route for that connection and its grid point is marked as unusable for future routes. If there is a minimum padding between connections required for fabrication reasons, then that number of additional units away from the route are also marked as unusable. This process is repeated for every connection in the system.

6.2 Control Layer Considerations

While control layer routing is beyond the scope of this article, Directed Placement does facilitate relaxation that can be useful when routing the control layer. Since Directed Placement places flow-layer components in a left-to-right orientation, it is advised that control layer I/O should be placed along the top and/or bottom edge of the device. From here, control lines can be routed through the buffer space between lanes or directly through components (where fabrication allows) to the edges. Pin insertion methods [17] could also use the interlane buffer space to insert control pins closer to the components that they control to reduce control route length.

In both these cases, the amount of unused space that can be utilized for control routing can be increased in a targeted manner through the manipulation of the lane buffer space Δ for a subset of lanes. If, for example, a component $c_j \in L_i$ was unable to be routed to a viable control pin, then the Δ between lanes L_{i-1}, L_i and L_i, L_{i+1} could be increased by some value σ to allow more space for pin insertion or control line routing. This increase of σ would then retrigger the in-lane placement and routing steps and another attempt by the control routing method to find a set of valid routes. This process could be performed iteratively unless a valid control routing was found or a maximum size threshold was reached.

7 RESULTS

The Directed Placement paired with the Lees-based router described here is compared to a Planar Placement algorithm paired with a Network Flow-based routing algorithm [29], a Simulated Annealing-based placer [32] paired with Hadlock's maze routing algorithm [35], and a Diagonal Component Expansion [6] algorithm paired with a Network Flow-based routing algorithm [6]. The Diagonal Component Expansion algorithm also includes the postprocessing method introduced in the same paper that utilizes the diagonal nature of the resulting layout to increase area utilization. All of these algorithms were implemented in C++ utilizing a *unitless* grid, which decouples the layout and design rule checking processes from the manufacturing resolution of any one specific mVLSI technology [33].

7.1 Benchmarks

We obtained netlists for four real-world planar mVLSI devices that have been designed, fabricated, and evaluated, as well as six netlists obtained by synthesizing synthetic benchmarks. The real-world netlists are as follows: **AquaFlex-3b and AquaFlex-5a** are proprietary mVLSI LoC netlists provided by Microfluidic Innovations, LLC; **HIV1** is a multilayer PDMS chip that performs a bead-based HIV1 p24 sandwich immunoassay [26]; and **MGG** is a molecular gradients generator that can generate five concentration levels of a two-sample mixture [42]. Five of the synthetic benchmarks (**Synthetic 1–5**) were generated by compiling a set of publicly available DAG specifications[1] through an established mVLSI architectural synthesis flow [31, 35]. The last

[1] https://sites.google.com/site/mlsibiochips/home.

	AquaFlex-3b	AquaFlex-5b	HIV1 [26]	MGG [42]	Synthetic 1	Synthetic 2	Synthetic 3	Synthetic 4	Synthetic 5	High Conn
# Components	15	17	13	30	21	15	34	34	46	24
# Connections	14	16	12	37	21	21	33	34	45	42

Fig. 10. The number of components and connections for each benchmark, Real Life on the top and Synthetic on bottom.

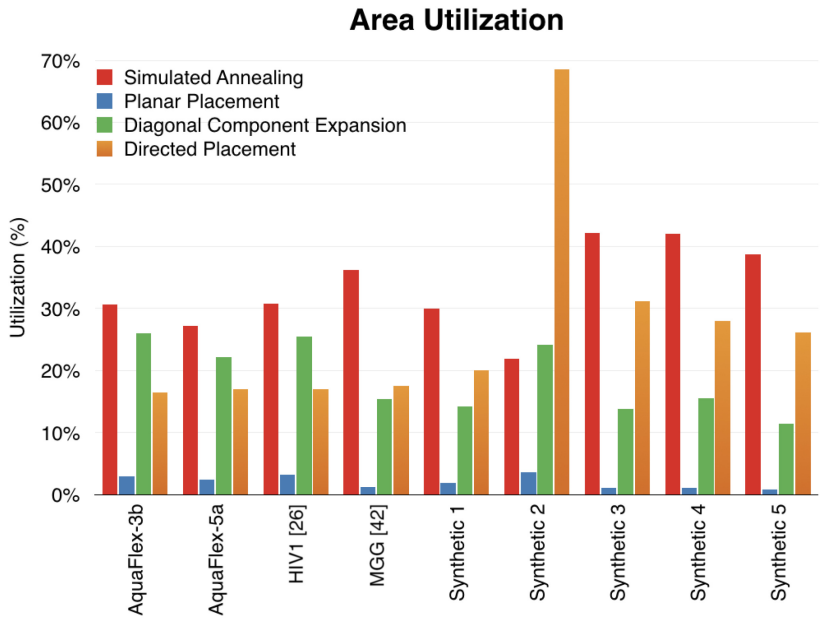

Fig. 11. The sum of the area of all the components in the device divided by the total area required to place and route the device, represented as a percentage per benchmark.

synthetic benchmark (**High Conn**) was designed to test high-connectivity devices where the average connectivity of a component is greatert than one, which often occurs in multiplexed systems such as fluidic storage [49]. The number of components and connections in each benchmark can be found in Figure 10.

7.2 Results and Analysis

For all benchmarks except **High Conn**, we report the area utilization (Figure 11: the ratio of component area to total chip area expressed as a percentage), average fluid channel length (Figure 12), average fluid channel length reduction (Figure 13), and average runtime (Figure 15). High Conn is an outlier and is discussed in more detail in Section 7.2.4. Directed Placement and Planar Placement achieved planar layouts for all benchmarks other than **High Conn**, while Simulated Annealing did not. As discussed earlier, Simulated Annealing is unlikely to generate planar layouts, even for planar netlists: we do not report the number of channel crossings in the layouts produced by Simulated Annealing, but the number was nonzero in all cases. Additionally, we remove the component

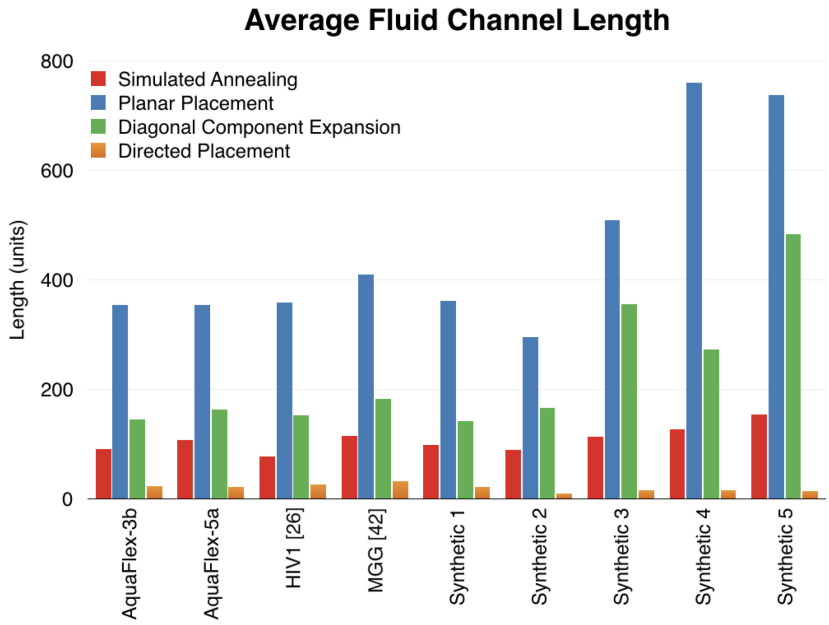

Fig. 12. The average length of all the fluid channels present in the device per benchmark.

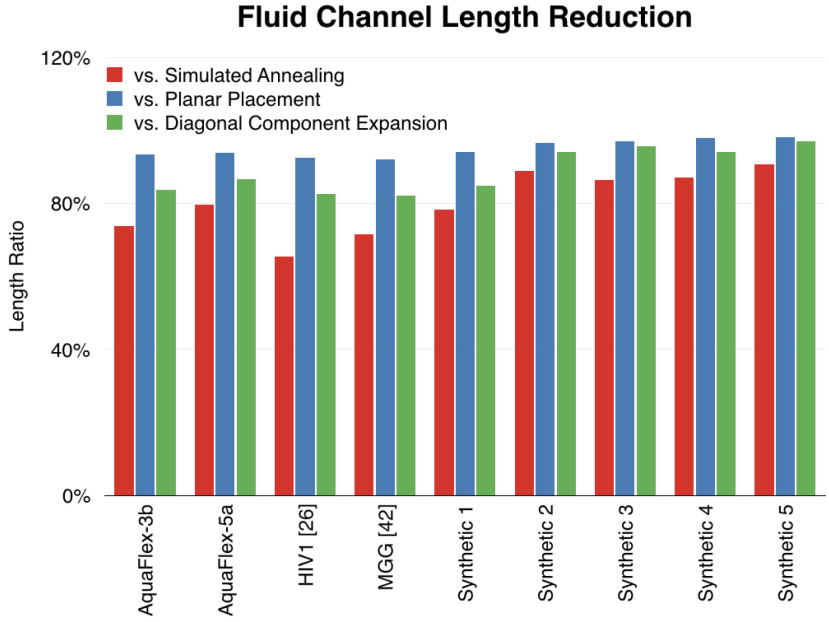

Fig. 13. The percent reduction in the average fluid channel route length when compared against Directed Placement per benchmark.

segmentation requirement from Simulated Annealing, which caused it to be tied very closely to its initial placement. The removal of the planar routing and the component segmentation requirements creates unrealistic designs but is a good point of comparison closer to the optimal.

7.2.1 Area Utilization. In Figure 11, Simulated Annealing achieves the highest area utilization for all the test cases except one. This result is expected since the Simulated Annealing method is primarily focused on optimizing the total area of the device and ignores the requirement that no routes intersect in the system. The one benchmark that Simulated Annealing is not best suited for is **Synthetic 2**. Directed Placement and Diagonal Component Expansion are especially effective on the **Synthetic 2** benchmark, increasing its area utilization from 22.65% with Simulated Annealing, 3.60% with Planar Placement, and 24.2% with Diagonal Component Expansion to 68.57% with Directed Placement. This dramatic increase on this particular benchmark is due to its particularly linear nature, yielding a straight-line layout with Directed Placement and a relatively linear placement in Diagonal Component Expansion. The rest of the benchmarks have a more complex placement and do not allow for this type of straight-line placement. On average, Directed Placement is 81.60% as effective as the Simulated Annealing method for area utilization.

7.2.2 Fluid Channel Length. For all benchmarks in Figure 12, Directed Placement achieves the shortest average fluid channel length. This is because Directed Placement utilizes the tree-like structure of mVLSI devices to create designs that try to optimize the placement of neighboring components as close as possible, as illustrated in Figure 14(d). By optimizing in this way, we are able to derive a large reduction in route length versus the other methods.

Planar Placement utilizes a planar embedding for its initial placement, which gives it a high probability of yielding a planar routing. However, the initial planar embedding tends to lay out the components into triangular substructures with increasing straight-line distances between them. This leads to small densely packed subgroups with large distances between them like those found in Figure 14(a). Additionally, the expansion method used in the Planar Placement method to avoid component intersections, while easing routing densities and further increasing the probability that a valid planar route can be found, also increases the necessary route distance between components.

Diagonal Component Expansion arranges the components across the diagonal of the layout without particular regard to their ordering. As can be seen in Figure 14(b), after the device is cropped along the diagonal and reoriented to create a small total device area, it has densely packed connections, some of which must traverse the majority of the device length. This leads to a similar routing situation as Planar Placement, with a mix of short and very long routes.

Simulated Annealing, in contrast to the above methods, starts with an initial placement and incrementally adjusts the result via random perturbation; while simulated annealing methods gained traction in standard cell placement in the mid-1980s [44, 45], the problem formulation was different in two key respects. First, the placement of each standard cell is characterized by a pair (i, j), indicating that the cell occupies position j in row i; as continuous flow microfluidics lacks standard cells, the position of each component is an (x, y) location in a 2D plane. Second, standard cell placement assumes multiple layers of metal for routing, which eliminates the requirement that the layout be planar; microfluidics, in contrast, imposes planarity as a requirement. Given these factors, Simulated Annealing struggles to identify perturbations that can simultaneously improve the layout quality while maintaining planarity; as such, it is notably ineffective in this context.

Directed Placement uses the same Network Flow-based routing algorithm as Planar Placement and Diagonal Component Expansion, which actively avoids the introduction of channel intersections into the systems, thereby ensuring planarity. Simulated Annealing, in contrast, optimizes route length but treats the number of intersections as one part of a multiobjective optimization function. While Simulated Annealing achieves shorter overall channel length than the alternatives,

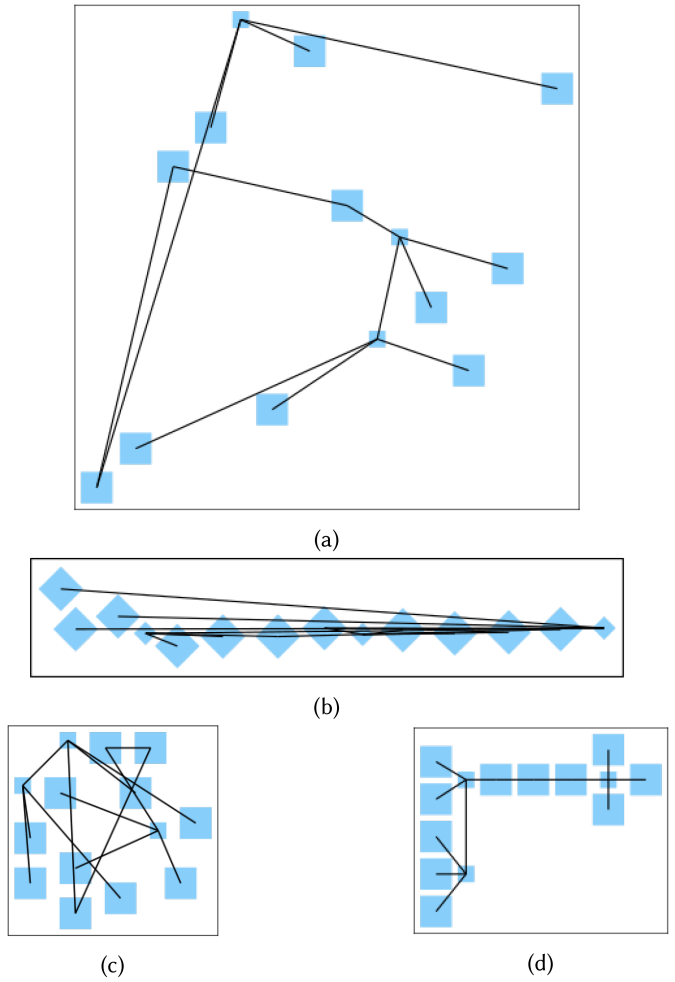

Fig. 14. Placements generated by (a) Planar Placement, (b) Diagonal Component Expansion, (c) Simulated Annealing, and (d) Directed Placement prior to routing. Blue bounding boxes represent placed components and black lines represent to-be-routed channels.

it does so at the cost of introducing additional channel intersections; this, in turn, necessitates additional control inputs, which may violate foundry design rules that impose a limit on the total number of inputs (both flow and control). As such, Simulated Annealing is an interesting baseline for comparison but, for all intents and purposes, is not a practical placement solution.

7.2.3 Runtime. Figure 15 shows the average runtime of each algorithm for each benchmark over five runs. Simulated Annealing has variable parameters that will affect both its runtime and the solution that it converges to. For the results presented here, Simulated Annealing was run with 100,000 moves per temperature change, a cooling rate of 1%, and an initial temperature of 100. When Directed Placement is compared to Planar Placement and Diagonal Component Expansion, Planar Placement and Diagonal Component Expansion tend to run faster on smaller benchmarks, while Directed Placement runs faster on larger ones. This occurs because the Directed Placement algorithm is more complex than Planar Placement and Diagonal Component Expansion but yields a better placement for the routing step. Since all three methods utilize the same or similar

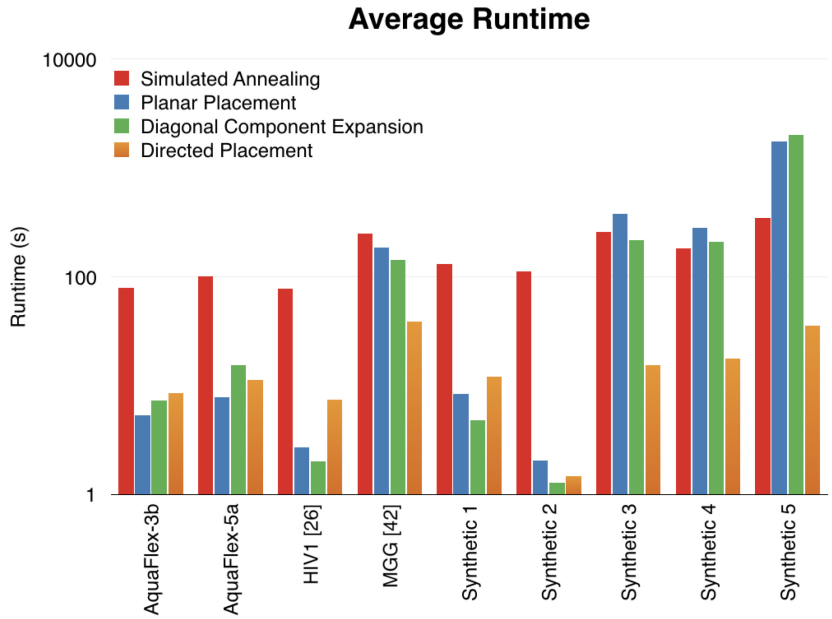

Fig. 15. The average runtime of all algorithms over five runs per benchmark.

routing steps, on small test cases where the routing makes up a small portion of the runtime, Planar Placements and Diagonal Component Expansion run faster, but as the routing requirements increase, Directed Placements' superior layout means a shorter routing time and a faster overall runtime. The one exception to this is the **Synthetic 2** benchmark, which runs fastest on Diagonal Component Expansion while still yielding a longer average fluid channel length. This is because the straight-line nature of that particular benchmark is trivial for the expansion method used in Diagonal Component Expansion and yields long fluid channels with few possibilities for intersections. Since the Network Flow-based router will perform a rip and reroute step if an intersection occurs, a reduction in possible intersections leads to a large reduction in the overall runtime. Because the Simulated Annealing method uses a Hadlocks/based router that does not avoid intersections, the vast majority of the time reported is spent in the placement stage. All other methods spend the majority of their time performing the routing step.

7.2.4 The High Conn Benchmark. The **High Conn** benchmark represents an anomaly when the results are compared to those of the other benchmarks. **High Conn** represents a device where a small number of components each have a large number of connections. With this benchmark, the Simulated Annealing method was able to find a valid placement and routing, but not without adding a nonzero number of additional intersections as it did with the other benchmarks. The Planar Placement and Diagonal Comment Expansion methods were able to generate valid placements but unable to find a valid routing. Connection-dense components represent a difficult problem for the Planar Placement and Directed Component Expansion methods because these components are not treated any differently from others in the system, leading to high congestion in portions of the layout and routing failures. Only the Directed Placement method was able to generate a valid placement and routing of the **High Conn** benchmark. With the Directed Placement method, the highly connected components are allocated to their own lanes, while the components they are connected to are all allocated into subsequent lane(s) and are subordered within their lane to try and eliminate cross-lane intersections, creating a much easier routing problem.

8 CONCLUSION AND FUTURE WORK

This article introduced Directed Placement, a new method for the placement and routing of mVLSI devices. This new method reduces fluid channel length at the cost of a small increase in the area utilization over previous heuristics. Additionally, the use of lanes and a straightforward left-to-right placement scheme yields layouts that are easier for designers to understand and modify, even at large scales. That being said, there is one key issue that requires further investigation.

It is still unclear what characteristics of a device layout will cause that device to function properly postfabrication. More investigation needs to take place into classifying different types of microfluidic devices and determining for each classification what features of the layout are important and to what degree, which will allow for placement and routing algorithms to be more rigorously validated. We plan to further investigate this issue in the future.

REFERENCES

[1] [n.d.]. Design Your Own Device: Basic Design Rules. Retrieved May 30, 2019, from https://web.stanford.edu/group/foundry/.

[2] Nada Amin, William Thies, and Saman P. Amarasinghe. 2009. Computer-aided design for microfluidic chips based on multilayer soft lithography. In *27th International Conference on Computer Design (ICCD'09)* 2–9. DOI: https://doi.org/10.1109/ICCD.2009.5413185

[3] Ismail Emre Araci and Stephen R. Quake. 2012. Microfluidic very large scale integration (mVLSI) with integrated micromechanical valves. *Lab-on-a-Chip* 12, 16 (Aug. 2012), 2803–2806. DOI: https://doi.org/10.1039/c2lc40258k

[4] Frederick K. Balagaddé, Lingchong You, Carl L. Hansen, Frances H. Arnold, and Stephen R. Quake. 2005. Long-term monitoring of bacteria undergoing programmed population control in a microchemostat. *Science (New York, N.Y.)* 309, 5731 (July 2005), 137–40. DOI: https://doi.org/10.1126/science.1109173

[5] H. Nelson Brady. 1984. An approach to topological pin assignment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 3, 3 (1984), 250–255. DOI: https://doi.org/10.1109/TCAD.1984.1270082

[6] Brian Crites, Karen Kong, and Philip Brisk. 2017. Diagonal component expansion for flow-layer placement of flow-based microfluidic biochips. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 5s (2017), 126.

[7] Dino Di Carlo, Nima Aghdam, and Luke P. Lee. 2006. Single-cell enzyme concentrations, kinetics, and inhibition analysis using high-density hydrodynamic cell isolation arrays. *Analytical Chemistry* 78, 14 (2006), 4925–4930. DOI: https://doi.org/10.1021/ac060541s

[8] Jamil El-Ali, Peter K. Sorger, and Klavs F. Jensen. 2006. Cells on chips. *Nature* 442, 7101 (2006), 403–411.

[9] Luis M. Fidalgo and Sebastian J. Maerkl. 2011. A software-programmable microfluidic device for automated biology. *Lab Chip* 11, 9 (2011), 1612–1619. DOI: https://doi.org/10.1039/C0LC00537A

[10] Hua Gong, Adam T. Woolley, and Gregory P. Nordin. 2016. High density 3D printed microfluidic valves, pumps, and multiplexers. *Lab Chip* 16, 13 (2016), 2450–2458. DOI: https://doi.org/10.1039/C6LC00565A

[11] Andreas Grimmer, Philipp Frank, Philipp Ebner, Sebastian Häfner, Andreas Richter, and Robert Wille. 2018. Meander designer: Automatically generating meander channel designs. *Micromachines* 9, 12 (2018), 625.

[12] Andreas Grimmer, Qin Wang, Hailong Yao, Tsung-Yi Ho, and Robert Wille. 2017. Close-to-optimal placement and routing for continuous-flow microfluidic biochips. In *22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. 530–535.

[13] William H. Grover, Alison M. Skelley, Chung N. Liu, Eric T. Lagally, and Richard A. Mathies. 2003. Monolithic membrane valves and diaphragm pumps for practical large-scale integration into glass microfluidic devices. *Sensors and Actuators B: Chemical* 89, 3 (2003), 315–323. DOI: https://doi.org/10.1016/S0925-4005(02)00468-9

[14] Jong Wook Hong and Stephen R. Quake. 2003. Integrated nanoliter systems. *Nature Biotechnology* 21, 10 (Oct. 2003), 1179–1183. DOI: https://doi.org/10.1038/nbt871

[15] Jong Wook Hong, Vincent Studer, Giao Hang, W. French Anderson, and Stephen R. Quake. 2004. A nanoliter-scale nucleic acid processor with parallel architecture. *Nature Biotechnology* 22 (March 2004), 435–439. https://doi.org/10.1038/nbt951

[16] Yi-Ling Hsieh and Tsung-Yi Ho. 2011. Automated physical design of microchip-based capillary electrophoresis systems. In *VLSI Design 2011: 24th International Conference on VLSI Design, IIT*. 165–170. DOI: https://doi.org/10.1109/VLSID.2011.47

[17] Kai Hu, Trung Anh Dinh, Tsung-Yi Ho, and Krishnendu Chakrabarty. 2017. Control-layer routing and control-pin minimization for flow-based microfluidic biochips. *IEEE Transactions on CAD of Integrated Circuits and Systems* 36, 1 (2017), 55–68. DOI: https://doi.org/10.1109/TCAD.2016.2568198

[18] Bo Huang, Hongkai Wu, Samuel Kim, and Richard N. Zare. 2005. Coating of poly(dimethylsiloxane) with n-dodecyl-Î²-d-maltoside to minimize nonspecific protein adsorption. *Lab Chip* 5, 10 (2005), 1005–1007. DOI: https://doi.org/10.1039/B509251E

[19] Paul J. Hung, Philip J. Lee, Poorya Sabounchi, Robert Lin, and Luke P. Lee. 2005. Continuous perfusion microfluidic cell culture array for high-throughput cell-based assays. *Biotechnology and Bioengineering* 89, 1 (2005), 1–8.

[20] E. C. Jensen, W. H. Grover, and R. A. Mathies. 2007. Micropneumatic digital logic structures for integrated microdevice computation and control. *Journal of Microelectromechanical Systems* 16, 6 (Dec. 2007), 1378–1385. DOI: https://doi.org/10.1109/JMEMS.2007.906080

[21] Xiran Jiang, Ning Shao, Wenwen Jing, Shengce Tao, Sixiu Liu, and Guodong Sui. 2014. Microfluidic chip integrating high throughput continuous-flow PCR and DNA hybridization for bacteria analysis. *Talanta* 122 (2014), 246–250. DOI: https://doi.org/10.1016/j.talanta.2014.01.053

[22] Casimir Kuratowski. 1930. Sur le problème des courbes gauches en Topologie. *Fundamenta Mathematicae* 15, 1 (1930), 271–283.

[23] Ali Lashkaripour, Christopher Rodriguez, Luis Ortiz, and Douglas Densmore. 2019. Performance tuning of microfluidic flow-focusing droplet generators. *Lab Chip* 19, 6 (2019), 1041–1053. DOI: https://doi.org/10.1039/C8LC01253A

[24] Ali Lashkaripour, Ryan Silva, and Douglas Densmore. 2018. Desktop micromilled microfluidics. *Microfluidics and Nanofluidics* 22, 3 (Feb. 2018), 31. DOI: https://doi.org/10.1007/s10404-018-2048-2

[25] C. Y. Lee. 1959. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers* 30 (1959), 1389–1401.

[26] Baichen Li, Lin Li, Allan Guan, Quan Dong, Kangcheng Ruan, Ronggui Hu, and Zhenyu Li. 2014. A smartphone controlled handheld microfluidic liquid handling system. *Lab-on-a-Chip* 14, 20 (2014), 4085–4092. DOI: https://doi.org/10.1039/C4LC00227J

[27] Chun-Xun Lin, Chih-Hung Liu, I-Che Chen, D. T. Lee, and Tsung-Yi Ho. 2014. An efficient bi-criteria flow channel routing algorithm for flow-based microfluidic biochips. In *Proceedings of the the 51st Annual Design Automation (DAC'14)*. 141:1–141:6. DOI: https://doi.org/10.1145/2593069.2593084

[28] Chunfeng Liu, Bing Li, Tsung-Yi Ho, Krishnendu Chakrabarty, and Ulf Schlichtmann. 2018. Design-for-testability for continuous-flow microfluidic biochips. In *Proceedings of the 55th Annual Design Automation Conference (DAC'18)*. 164:1–164:6. DOI: https://doi.org/10.1145/3195970.3196025

[29] Jeffrey McDaniel, Auralila Baez, Brian Crites, Aditya Tammewar, and Philip Brisk. 2013. Design and verification tools for continuous fluid flow-based microfluidic devices. In *Asia and South Pacific Design Automation Conference (ASPDAC'13)*.

[30] Jeffrey McDaniel, Brian Crites, Philip Brisk, and William H. Grover. 2015. Flow-layer physical design for microchips based on monolithic membrane valves. *IEEE Design & Test* 32, 6 (2015), 51–59. DOI: https://doi.org/10.1109/MDAT.2015.2459699

[31] Jeffrey McDaniel, Christopher Curtis, and Philip Brisk. 2013. Automatic synthesis of microfluidic large scale integration chips from a domain-specific language. In *Proceedings of the IEEE Biomedical Circuits and Systems Conference (BioCAS'13)*. 101–104.

[32] Jeffrey McDaniel, Brendon Parker, and Philip Brisk. 2014. Simulated annealing-based placement for microfluidic large scale integration (mLSI) chips. In *Proceedings of the 22nd IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC'14)*. 213–218.

[33] Carver Mead and Lynn Conway. 1980. *Introduction to VLSI Systems*. Addison-Wesley.

[34] Jessica Melin and Stephen R. Quake. 2007. Microfluidic large-scale integration: The evolution of design rules for biological automation. *Annual Review of Biophysics and Biomolecular Structure* 36 (Jan. 2007), 213–231. DOI: https://doi.org/10.1146/annurev.biophys.36.040306.132646

[35] Wajid Hassan Minhass, Paul Pop, Jan Madsen, and Felician Stefan Blaga. 2012. Architectural synthesis of flow-based microfluidic large-scale integration biochips. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis of Embedded Systems (CASES'12)*. 181–190.

[36] Wajid Hassan Minhass, Paul Pop, Jan Madsen, and Tsung-yi Ho. 2013. Control synthesis for the flow-based microfluidic large-scale integration biochips. In *Asia and South Pacific Design Automation Conference (ASPDAC'13)*.

[37] Shirin Mesbah Oskui, Graciel Diamante, Chunyang Liao, Wei Shi, Jay Gan, Daniel Schlenk, and William H. Grover. 2016. Assessing and reducing the toxicity of 3D-printed parts. *Environmental Science & Technology Letters* 3, 1 (2016), 1–6. DOI: https://doi.org/10.1021/acs.estlett.5b00249

[38] Nicole Pamme. 2007. Continuous flow separations in microfluidic devices. *Lab on a Chip* 7, 12 (2007), 1644–1659.

[39] Anton J. Pfeiffer, Tamal Mukherjee, and Steinar Hauan. 2006. Synthesis of multiplexed biofluidic microchips. *IEEE Transactions on CAD of Integrated Circuits and Systems* 25, 2 (2006), 321–333. DOI: https://doi.org/10.1109/TCAD.2005.855931

[40] Seetal Potluri, Paul Pop, and Jan Madsen. 2019. Design-for-testability of on-chip control in mVLSI biochips. *IEEE Design & Test* 36, 1 (2019), 48–56. DOI: https://doi.org/10.1109/MDAT.2018.2873448

[41] Seetal Potluri, Alexander Schneider, Martin Horslev-Petersen, Paul Pop, and Jan Madsen. 2017. Synthesis of on-chip control circuits for mVLSI biochips. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*. 1799–1804. DOI: https://doi.org/10.23919/DATE.2017.7927284

[42] Minsoung Rhee and Mark A. Burns. 2008. Microfluidic assembly blocks. *Lab-on-a-Chip* 8, 8 (2008), 1365–1373. DOI: https://doi.org/10.1039/B805137B

[43] Chad I. Rogers, Kamran Qaderi, Adam T. Woolley, and Gregory P. Nordin. 2015. 3D printed microfluidic devices with integrated valves. *Biomicrofluidics* 9, 1 (2015), 016501. DOI: https://doi.org/10.1063/1.4905840

[44] C. Sechen and A. Sangiovanni-Vincentelli. 1985. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits* 20, 2 (April 1985), 510–522. DOI: https://doi.org/10.1109/JSSC.1985.1052337

[45] Carl Sechen and Alberto L. Sangiovanni-Vincentelli. 1986. TimberWolf3.2: A new standard cell placement and global routing package. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*. 432–439. DOI: https://doi.org/10.1145/318013.318083

[46] Jayna J. Shah, Jon Geist, Laurie E. Locascio, Michael Gaitan, Mulpuri V. Rao, and Wyatt N. Vreeland. 2006. Surface modification of poly(methyl methacrylate) for improved adsorption of wall coating polymers for microchip electrophoresis. *ELECTROPHORESIS* 27, 19 (2006), 3788–3796. DOI: https://doi.org/10.1002/elps.200600118

[47] Taraneh Taghavi, Xiaojian Yang, Bo-Kyung Choi, Maogang Wang, and Majid Sarrafzadeh. 2006. Dragon2006: Blockage-aware congestion-controlling mixed-size placer. In *Proceedings of the 2006 International Symposium on Physical Design (ISPD'06)*. 209–211. DOI: https://doi.org/10.1145/1123008.1123054

[48] S. C. Terry, J. H. Jerman, and J. B. Angell. 1979. A gas chromatographic air analyzer fabricated on a silicon wafer. *IEEE Transactions on Electron Devices* 26, 12 (Dec. 1979), 1880–1886. DOI: https://doi.org/10.1109/T-ED.1979.19791

[49] Todd Thorsen, Sebastian J. Maerkl, and Stephen R. Quake. 2002. Microfluidic large-scale integration. *Science* 298, 5593 (2002), 580–584.

[50] Kai-Han Tseng, Sheng-Chi You, Jhe-Yu Liou, and Tsung-Yi Ho. 2013. A top-down synthesis methodology for flow-based microfluidic biochips considering valve-switching minimization. In *Proceedings of the International Symposium on Physical Design (ISPD'13)*. 123–129.

[51] Tsun-Ming Tseng, Mengchu Li, Daniel Nestor Freitas, Travis McAuley, Bing Li, Tsung-Yi Ho, Ismail Emre Araci, and Ulf Schlichtmann. 2018. Columba 2.0: A co-layout synthesis tool for continuous-flow microfluidic biochips. *IEEE Transactions on CAD of Integrated Circuits and Systems* 37, 8 (2018), 1588–1601. DOI: https://doi.org/10.1109/TCAD.2017.2760628

[52] Tsun-Ming Tseng, Mengchu Li, Daniel Nestor Freitas, Amy Mongersun, Ismail Emre Araci, Tsung-Yi Ho, and Ulf Schlichtmann. 2018. Columba S: A scalable co-layout design automation tool for microfluidic large-scale integration. In *Proceedings of the 55th Annual Design Automation Conference (DAC'18)*. 163:1–163:6. DOI: https://doi.org/10.1145/3195970.3196011

[53] Tsun-Ming Tseng, Mengchu Li, Bing Li, Tsung-Yi Ho, and Ulf Schlichtmann. 2016. Columba: Co-layout synthesis for continuous-flow microfluidic biochips. In *Proceedings of the 53rd Annual Design Automation Conference (DAC'16)*. 147:1–147:6. DOI: https://doi.org/10.1145/2897937.2897997

[54] Marc Alexander Unger, Hou-Pu Chou, Todd Thorsen, Axel Scherer, and Stephen R. Quake. 2000. Monolithic microfabricated valves and pumps by multilayer soft lithography. *Science* 288, 5463 (April 2000), 113–116. DOI: https://doi.org/10.1126/science.288.5463.113

[55] John Paul Urbanski, William Thies, Christopher Rhodes, Saman Amarasinghe, and Todd Thorsen. 2006. Digital microfluidics using soft lithography. *Lab-on-a-Chip* 6, 1 (2006), 96–104. DOI: https://doi.org/10.1039/B510127A

[56] Sidra Waheed, Joan M. Cabot, Niall P. Macdonald, Trevor Lewis, Rosanne M. Guijt, Brett Paull, and Michael C. Breadmore. 2016. 3D printed microfluidic devices: Enablers and barriers. *Lab Chip* 16, 11 (2016), 1993–2013. DOI: https://doi.org/10.1039/C6LC00284F

[57] Junchao Wang, Philip Brisk, and William H. Grover. 2016. Random design of microfluidics. *Lab on a Chip* 16, 21 (2016), 4212–4219.

[58] Qin Wang, Hao Zou, Hailong Yao, Tsung-Yi Ho, Robert Wille, and Yici Cai. 2018. Physical co-design of flow and control layers for flow-based microfluidic biochips. *IEEE Transactions on CAD of Integrated Circuits and Systems* 37, 6 (2018), 1157–1170. DOI: https://doi.org/10.1109/TCAD.2017.2748003

[59] Richard A. White, Paul C. Blainey, H. Christina Fan, and Stephen R. Quake. 2009. Digital PCR provides sensitive and absolute calibration for high throughput sequencing. *BMC Genomics* 10, 1 (March 2009), 116. DOI: https://doi.org/10.1186/1471-2164-10-116

[60] Angela R. Wu, Tiara L. A. Kawahara, Nicole A. Rapicavoli, Jan van Riggelen, Emelyn H. Shroff, Liwen Xu, Dean W. Felsher, Howard Y. Chang, and Stephen R. Quake. 2012. High throughput automated chromatin immunoprecipitation as a platform for drug screening and antibody validation. *Lab on a Chip* 12, 12 (June 2012), 2190–2198. DOI: https://doi.org/10.1039/c2lc21290k

[61] Younan Xia and George M. Whitesides. 1998. Soft lithography. *Annual Review of Materials Science* 28, 1 (1998), 153–184.

[62] Hailong Yao, Tsung-Yi Ho, and Yici Cai. 2015. PACOR: Practical control-layer routing flow with length-matching constraint for flow-based microfluidic biochips. In *Proceedings of the 52nd Annual Design Automation Conference*. 142:1–142:6. DOI: https://doi.org/10.1145/2744769.2744887

[63] Hailong Yao, Qin Wang, Yizhong Ru, Yici Cai, and Tsung-Yi Ho. 2015. Integrated flow-control codesign methodology for flow-based microfluidic biochips. *IEEE Design & Test* 32, 6 (2015), 60–68. DOI: https://doi.org/10.1109/MDAT.2015.2449180

[64] Ying Zhu, Bing Li, Tsung-Yi Ho, Qin Wang, Hailong Yao, Robert Wille, and Ulf Schlichtmann. 2018. Multi-channel and fault-tolerant control multiplexing for flow-based microfluidic biochips. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'18)*. 123. DOI: https://doi.org/10.1145/3240765.3240830

Received July 2018; revised September 2019; accepted October 2019